For this lab, you will practice writing loops. Refer to the lecture notes from Monday (the problems are similar to what we have done), and do as many problems as you can.

Problem 1. π

The following is the Leibniz formula for π:

$$\sum_{0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4}.$$

Write a program (using a `for`- loop) to compute

$$\sum_{0}^{1000} \frac{(-1)^n}{n} 2n + 1,$$

and then print an approximation for π using the result of the computation.

See `https://en.wikipedia.org/wiki/Leibniz_formula_for_%CF%80` for details on why this works. Hint: the following code computes

$$\sum_{0}^{500} n = 0 + 1 + 2 + 3 + ... + 500 :$$

```
res = 0
for n in range(501): # the last number to be added in should be 500
      res = res + n
```

For this problem, you are doing the same thing, but for a more complicated summand.

Problem 2. A bit more π

Now, write a program using a `while`-loop to compute the quantity from Problem 1. See last week's Lecture 3 ar around 20min for a technique for converting `for`-loops into `while`-loops.

Problem 3. Greatest common divisor using exhaustive search

In this question, you will write a function with the signature `gcd(n, m)` which computes the greatest common divisor of the positive integers `n` and `m`.

The greatest common divisor of n and m is a number d such that n is divisible by d **and** m is divisible by d.

Note that k is divisible by d if and only if the remainder of the division of k by d is 0, i.e., `k % d == 0`.

There is an efficient algorithm for doing finding the greatest common divisor (more on this later), but in this question, you should use *exhaustive search*: trying every possible answer until you find the right one. This is a similar approach to the one we used to compute $\log_{10}(n)$ in lecture.

Implement both of the approaches below.

Part (a) Approach 1

Write the function by trying every possible divisor from 1, 2, 3, ..., etc. (What is the largest guess you can try?). Keep track of the largest guess for the greatest common divisor that worked so far. Update a variable every time the divisor divides both `n` and `m`. Return the latest guess that worked.

Part (b) Approach 2

Approach one is inefficient in that we always have to try *all the possible guesses* every time. If we tried the largest guess first and it worked, we would not need to try smaller guesses (explain why).

Use a `while`-loop to try all the possible guesses, from the largest to the smallest, and use an early return technique to return from the function once you know the answer.

Hint: in the examples so far, we used something like this to count from 1 to n:

```
i = 1
while i < n:
      print(i)
      i = i + 1
```

Now, we want to count backward from a larger number to a smaller number. That means we want to change `i = 1` and `i = i + 1` to something else.

Note: it is also possible, and arguably better, to use a `for`-loop here (using material not covered until Week 4), but for this question, please use a `while`-loop.

Problem 4. Simplifying Fractions

Write a function with the signature `simpify_fraction(n, m)` which prints the simplified version of the fraction $\frac{n}{m}$.

You do not need to use a complicated algorithm to compute the greatest common divisor (although you certainly can do that!). For example, `simplify_fraction(3,6)` should print `1/2`, and

`simplify_fraction(8, 4)` should print `2`.

Problem 5. Sum more π

Now we would like to figure out how many terms in the summation above we need to add up in order to approximate π to `n` significant digits. Write a function that returns the number of the terms required to obtain an approximation of π using the Leibniz formula that agrees with the actual value of π to n significant digits (for the purposes of this problem, the approximation and π agree to n significant digits if the first n digits are the same in π and in the approximation.)

The best approximation of π using a `float` is available in `math.pi` (execute `import math` to be able to use it.)

Part of your job is to figure out whether two numbers agree to n significant digits. To figure that out, for a float `x`, consider what `int(x*(10**n))` means.

Problem 6. Calendar

Only do this problem after your work has been checked by a TA.

Part (a) Tomorrow's Date

Write a function with the signature `next_day(y, m , d)` which prints the date that follows the date `y/m/d`.

Reminder:

> According to the Gregorian calendar, which is the civil calendar in use today, years evenly divisible by 4 are leap years, with the exception of centurial years that are not evenly divisible by 400. Therefore, the years 1700, 1800, 1900 and 2100 are not leap years, but 1600, 2000, and 2400 are leap years. *(Source: the US Naval Observatory website.)*

Part (b) Counting Days

Write a function that prints out, in order, all the dates between `fY/fM/fD` and `tY/tM/tD`. Using the same idea, write a function that returns the *number* of days between two dates.

Problem 7. Euclid's Algorithm

Advanced problem. Only do this problem after your work has been checked by a TA.

A more efficient way of simplifying fractions than what we suggested for Problem 4 is Euclid's algorithm: `https://crypto.stanford.edu/pbc/notes/numbertheory/euclid.html`. Implement and test Euclid's algorithm.

You can and should reformulate the algorithm to use a `while`-loop.